UPF Library Preparation Application Notes

Version B-2008.09-SP3, March 2009

SYNOPSYS®

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Astro, Behavior Extracting Synthesis Technology, Cadabra, CATS, Certify, CHIPit, Design Compiler, DesignWare, Formality, HDL Analyst, HSIM, HSPICE, Identify, iN-Phase, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Physical Compiler, PrimeTime, SCOPE, Simply Better Results, SiVL, SNUG, SolvNet, Syndicated, Synplicity, Synplify, Synplify Pro, Synthesis Constraints Optimization Environment, TetraMAX, the Synplicity logo, UMRBus, VCS, Vera, and YIELDirector are registered trademarks of Synopsys, Inc.

Trademarks (™)

AFGen, Apollo, Astro-Rail, Astro-Xtalk, Aurora, AvanWaves, BEST, Columbia, Columbia-CE, Confirma, Cosmos, CosmosLE, CosmosScope, CRITIC, CustomSim, DC Expert, DC Professional, DC Ultra, Design Analyzer, Design Vision, DesignerHDL, DesignPower, DFTMAX, Direct Silicon Access, Discovery, Eclypse, Encore, EPIC, Galaxy, Galaxy Custom Designer, HANEX, HAPS, HapsTrak, HDL Compiler, Hercules, Hierarchical Optimization Technology, High-performance ASIC Prototyping System, HSIM^{plus}, i-Virtual Stepper, IICE, in-Sync, iN-Tandem, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, Liberty, Libra-Passport, Library Compiler, Magellan, Mars, Mars-Rail, Mars-Xtalk, Milkyway, ModelSource, Module Compiler, MultiPoint, Physical Analyst, Planet, Planet-PL, Polaris, Power Compiler, Raphael, Saturn, Scirocco, Scirocco-i, Star-RCXT, Star-SimXT, System Compiler, System Designer, Taurus, TotalRecall, TSUPREM-4, VCS Express, VCSi, VHDL Compiler, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (SM)

MAP-in, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
ARM and AMBA are registered trademarks of ARM Limited.
Saber is a registered trademark of SabreMark Limited Partnership and is used under license.
All other product or company names may be trademarks of their respective owners.

Contents

Preface

This preface includes the following sections:

- About These Application Notes
- Customer Support

About These Application Notes

The *UPF Library Preparation Application Notes* document provides Unified Power Format (UPF) library preparation guidelines and instructions for generating a UPF-ready library. It also describes how to update the port type of secondary power and ground (PG) pins in a FRAM-based view with information from a .db library. This document contains the following application notes:

- *Adding PG Pin Syntax to Logical Libraries Application Note*

  This application note provides UPF library preparation guidelines and instructions for generating a UPF-ready library, which is defined as any Liberty library that has been updated using Liberty power and ground (PG) pin syntax. This document describes how to use the `add_pg_pin_to_lib` command to convert and validate libraries that are not based on PG pin syntax to PG pin-based logical libraries.

- *Updating Secondary PG Pins Application Note*

  This application note describes how to update the port type of secondary power and ground (PG) pins in a FRAM-based view with information from a .db library. In many cases, the FRAM view in the library is missing secondary PG pin information. The port type must be updated to allow Milkyway tools to complete routing.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates command syntax.
`Courier italic`	Indicates a user-defined value in Synopsys syntax, such as *`object_name`*. (A user-defined value that is not Synopsys syntax, such as a user-defined value in a Verilog or VHDL statement, is indicated by regular text font italic.)
`Courier bold`	Indicates user input—text you type verbatim—in Synopsys syntax and examples. (User input that is not Synopsys syntax, such as a user name or password you enter in a GUI, is indicated by regular text font bold.)
[ ]	Denotes optional parameters, such as *`pin1 [pin2 ... pinN]`*
|	Indicates a choice among alternatives, such as `low | medium | high` (This example indicates that you can enter one of three possible values for an option: low, medium, or high.)
_	Connects terms that are read as a single term by the system, such as `set_annotated_delay`
Control-c	Indicates a keyboard combination, such as holding down the Control key and pressing c.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services including software downloads, documentation on the Web, and "Enter a Call to the Support Center."

To access SolvNet, go to the SolvNet Web page at the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.

If you need help using SolvNet, click HELP in the top-right menu bar or in the footer.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a call to your local support center from the Web by going to https://solvnet.synopsys.com (Synopsys user name and password required), and then clicking "Enter a Call to the Support Center."
- Send an e-mail message to your local support center.
  - E-mail support_center@synopsys.com from within North America.
  - Find other local support center e-mail addresses at http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - Call (800) 245-8005 from within the continental United States.
  - Call (650) 584-4200 from Canada.
  - Find other local support center telephone numbers at http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

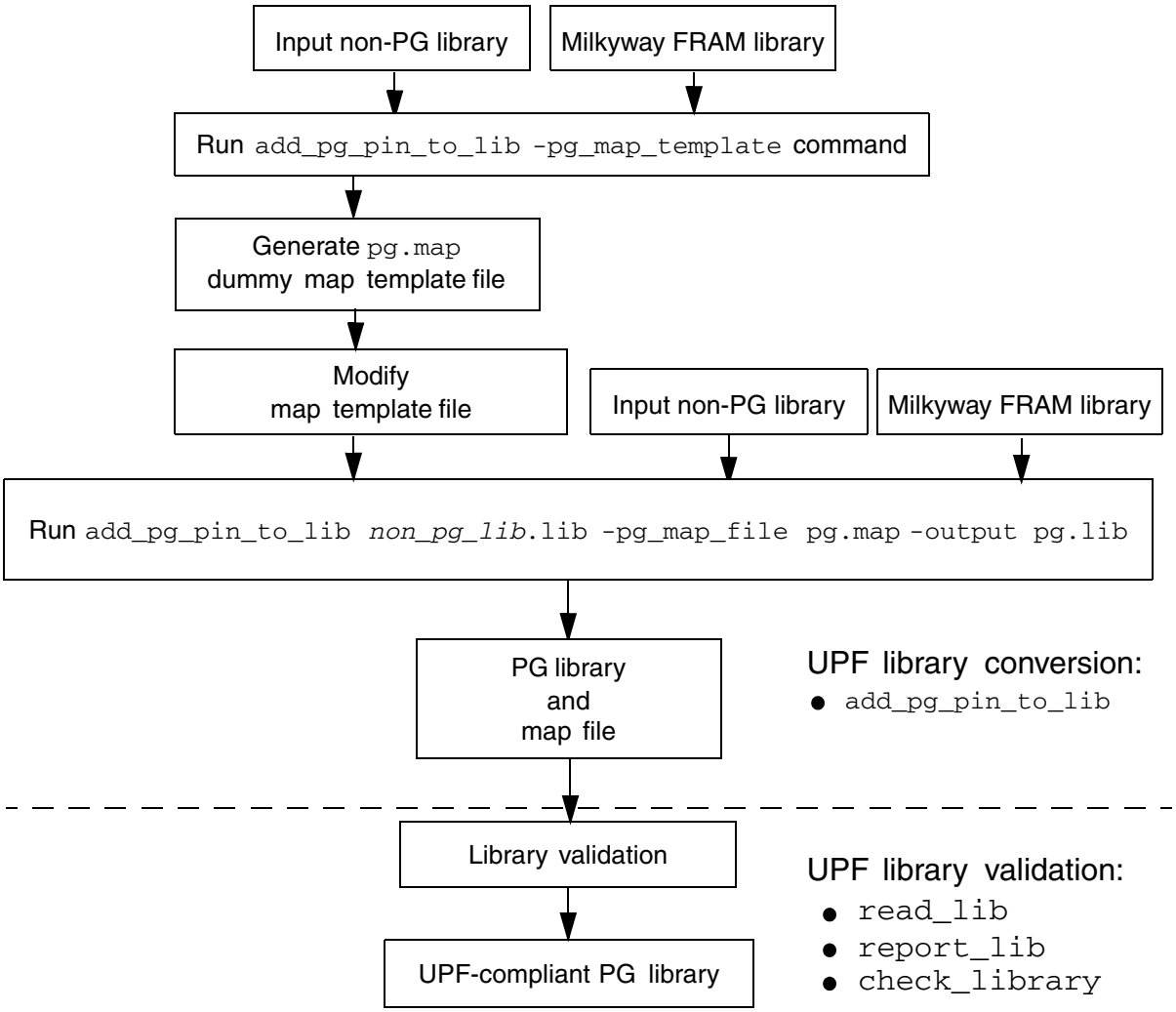
Adding PG Pin Syntax to Logical Libraries Application Note

This application note provides Unified Power Format (UPF) library preparation guidelines and instructions for generating a UPF-ready library. A UPF-ready library is defined as any Liberty library that has been updated using Liberty power and ground (PG) pin syntax. This document describes how to use the `add_pg_pin_to_lib` command to convert and validate libraries that are not based on PG pin syntax to PG pin-based logical libraries. This application note contains the following sections:

Generating Libraries With PG Pin Syntax

You can convert a library that is not based on PG pin syntax to a library with PG pin syntax by using the `add_pg_pin_to_lib` command. The converted library is UPF ready. You can then validate the library using library validation commands. Figure 1-1 highlights the steps that are necessary to convert a library using `add_pg_pin_to_lib` and validate the library using library validation commands.

Figure 1-1 Flow for Converting and Validating PG Pin Libraries



The `add_pg_pin_to_lib` command is available on the Library Compiler, Design Compiler, and IC Compiler command prompt. The command converts and updates a logical library automatically from the old `rail_connection` syntax or from a format that is not based on PG pin syntax to a library with PG pin syntax. You must input the logical library that is not based on PG pin syntax and the related physical libraries with a Milkyway FRAM view, if available, and also a PG map file (side file).

Note:

The `add_pg_pin_to_lib` command is the same as the `add_pg_pin_to_db` command except that the library file that you input when you use `add_pg_pin_to_lib` is a .lib ASCII file whereas the input file for the `add_pg_pin_to_db` command is a .db binary source file. The `add_pg_pin_to_db` command has been available since the A-2007.12 release and the automatic PG map template generation feature has been available since the A-2007.12-SP4 release. In all other ways, the `add_pg_pin_to_db` and `add_pg_pin_to_lib` commands have the same options and values and use the same flows.

add_pg_pin_to_lib Syntax

The following section describes the `add_pg_pin_to_lib` command syntax:

```
add_pg_pin_to_lib input_lib_filename
 [-mw_library_name mw_lib_name]
 [-pg_map_file pg_map_filename.map]
 [-pg_map_template pg.map_template_filename]
 [-expanded]
 [-verbose]
 [-common_shell_path common_shell_path]
-output pg_lib_filename
```

Arguments

input_lib_filename

Specifies the name of the input logical library (.lib) file that is not based on PG pin syntax. You must specify the input logical library file to run the `add_pg_pin_to_lib` command. If the `search_path` environment variable is set, only the file name is required; otherwise, you must specify a file name with the full path.

`-mw_library_name` *mw_lib_name*

Specifies one or more Milkyway library names or FRAM views that correspond to the input logical .db file. You must specify either the `-mw_library_name` option or the `-pg_map_file` option for single rail cells. If the cells in the library are multiple-rail cells, you must specify both the `-mw_library_name` and the `-pg_map_file` options. If the `search_path` environment variable is set, only the file name is required; otherwise, you must specify a file name with the full path.

`-pg_map_file` *`pg_map_filename.map`*

Specifies the file name for mapping between data that is not based on PG pin syntax and data that is based on PG pin syntax. You must specify either the `-pg_map_file` option or the `-mw_library_name` option for single rail cells. If the cells in the library are multiple-rail cells, you must specify both the `-mw_library_name` and `-pg_map_file` options. If you specify the map file name only, the map file is generated in the current working directory. To save it to another location, you must specify the full path to the map file location.

For information about the PG map file format, see "PG Map File Format" on page 1-5.

`-pg_map_template` *`pg.map_template_filename`*

Specifies the file name for the PG map file to be automatically generated. You can use this option if you want to quickly generate a PG map file template to use as the first step for building the final PG map file. Later, you can use this PG map file template to complete the final conversion step with the `-pg_map_file` option. When you specify the `-pg_map_template` option, you cannot use the `-pg_map_file` option during the same run.

If you specify the map file name only, the map file is generated in the current working directory. To save it to another location, you must specify the full path to the map file location.

You can use the `-pg_map_template` option with the input non-PG library file only or with the `-mw_library_file` option with the Milkyway FRAM view. It is recommended that you include Milkyway FRAM views during map template generation so that the `add_pg_pin_to_lib` command can generate a nearly complete PG map file. This is because PG information that is missing from the logical library can be derived from the Milkyway views during PG map file generation, creating a more complete PG map file.

For information about the PG map file format, see "PG Map File Format" on page 1-5.

`-expanded`

By default, the `add_pg_pin_to_lib` command tries to generate a PG map file that has as many wildcards as possible in compressed format in order to reduce the file size. However, if you want to debug the PG map file for any issues, you can use the `-expanded` option to generate a PG map file with all the wildcards expanded.

Note:

The `-expanded` option is available only with `add_pg_pin_to_lib` in dc_shell, lc_shell, and icc_shell, beginning with the B-2008.09-SP3 release.

For information about the PG map file format, see "PG Map File Format" on page 1-5.

`-verbose`

Outputs all sections of the PG map data in a log file, whether the PG map data is generated automatically or expanded from the map file.

`-common_shell_path` *`common_shell_path`*

Specifies a common_shell path where lc_shell or dc_shell resides. Use the `-common_shell_path` option if you want to use a specific version of the common_shell executable for the library compilation phase.

`-output` *`pg_db_filename`*

Specifies the name of the new PG pin-based file that is generated after the `add_pg_pin_to_lib` conversion. You must specify the file name. No default name is used for the file. If you specify the file name only and not the full path, the file is generated in the current working directory. To save it to another location, you must specify the full path.

Examples

In the following example, a logical library named `old_non_pg.lib` and a Milkyway library named `mw_lib` generate a PG map file template named `pg.map_dummy` in the current working directory:

```
command_shell> add_pg_pin_to_lib old_non_pg.lib \
-mw_library_name {mw_lib}                       \
-pg_map_template pg.map_dummy
```

Note:

The `-expanded` option is not specified, so a compressed PG map file template is created in the current working directory.

In the following example, the `old_non_pg.lib` logical library is converted to a PG pin-based library named `pg.lib`, which is written to the current working directory:

```
command_shell> add_pg_pin_to_lib old_non_pg.lib \
-mw_library_name {mw_lib}                       \
-pg_map_file pg.map                            \
-output pg.lib
```

PG Map File Format

The PG map file includes one or more sections for different groups of mapping, such as mapping between signal pins and related power and ground pins, mapping a PG pin to a voltage name and mapping the voltage name to a voltage value. Each section is enclosed by a `BEGIN` keyword and ends with an `END` keyword in the following order:

```
BEGIN section_name                                  <------ Section
field_1      field_2    field_3    field_n           <------ Title line
value1_1     value1_2   value1_3   value1_n
.....................
END section_name
```

Each section after the `BEGIN section_name` line should have a title line. The title line contains field names, which are keywords that you can separate by one or more spaces or Tabs. After the title line, you can specify values separated by one or more spaces or Tabs. All fields in the PG map file must be specified with values or special characters, such as an asterisk (*), a hyphen (-), and a caret (^). Leading spaces in each line are ignored after the file is read.

If the value in any section is the same as the value immediately above it, you can use a single caret (^) instead of duplicating the data. If no value is needed or if there is no need to input any value for any field, you can use a hyphen (-). You can leave the last field blank. The new-line character will fill the blanks with a hyphen (-).

You can use the asterisk (*) wildcard character to group multiple entries and specify partial names. For example, if you specify `AND*` under the cell field, the string `AND*` replaces cell names such as `AND1`, `AND2`, and so on. In another example, if you specify `AND1` as a cell name in the cell column and you specify the asterisk (*) wildcard character for the other cells, the command interprets the asterisk (*) character to mean all cells except `AND1`.

You can add comment lines anywhere in the PG map file using the hash character (#) at the beginning of the line. A comment line must be a complete line and should not be specified after a character and before the new-line character. The PG map file is order-independent so you can specify the sections in any order.

There are five sections in the PG map file. Each section has a unique section name and field names. The following sections describe the PG map file sections.

PG Pin-Mapping Format

Mapping between signal pins and related power and ground pins is called PG pin mapping. The following section shows the PG pin-mapping structure:

```
BEGIN PG_PIN_MAP
cell        pin                 rail_connection        pg_pin
cell_name  signal_pin_name    rail_connection_name     pg_pin_name
....................
END PG_PIN_MAP
```

`PG_PIN_MAP` is the section name for the signal-to-PG pin-mapping section in the PG map file. The title line contains four field names in the following order:

```
cell  pin  rail_connection pg_pin
```

The field names are separated by one or more spaces or Tabs. The `cell` field specifies the cell name, the `pin` field specifies the signal pin name, the `rail_connection` field specifies the power pin names, and the `pg_pin` field specifies the `related_power_pin` or `related_ground_pin` name in the pin group.

Use the following syntax to specify the `rail_connection` field:

```
rail_connection(rc_name, rail_name);
```

Note:
Make sure you specify the fields in exactly the same order that is specified in the title line.

The mapping values follow the title line. The values have the same number of columns as the title keywords and are separated by one or more spaces or Tabs, as shown:

```
cell_name signal_pin_name rail_connection_name pg_pin_name
```

The following example shows an input PG map file for a single rail:

```
#
# pg_map_file for single rail
#
BEGIN PG_PIN_MAP
cell        pin  rail_connection  pg_pin
LS_1        A    P1               VDD1
^           ^    -                VSS1
END PG_PIN_MAP

#
```

The following example shows an input PG map file for multiple rails:

```
# pg_map_file for multiple rails
#
BEGIN PG_PIN_MAP
cell        pin  rail_connection    pg_pin
LS_1        A    P1                 VDD
^           ^    -                  VSS
^           Z    P2                 VDD1
^           ^    -                  VSS1
END PG_PIN_MAP
#
```

The following example shows a map file for a standard cell library containing only single rail cells:

```
BEGIN PG_PIN_MAP
cell        pin  rail_connection   pg_pin
*           *    -                 VDD
```

```
*           *    -                  VSS
END PG_PIN_MAP
```

The `pg_pin` names in the PG map file should match the PG pin names in the Milkyway library exactly. A PG map file is required for libraries with multiple-power rail cells.

Use the hyphen (-) when no value is needed or if there is no need to input any value. For example, if the input library file has no rail information, `rail_connection` has no value in the map file. Therefore, you should specify a hyphen (-) as the value. Do not leave the field blank.

PG Pin-to-Voltage Name Mapping Section

You can specify the PG pin-to-voltage name and PG-type mapping information in the PG map file in the `PG_TO_VOLTAGE_MAP` section.

The following section shows the PG pin-to-voltage name and the pg_type mapping structure:

```
BEGIN PG_TO_VOLTAGE_MAP
cell         pg_pin         voltage_name     pg_type    direction
cell_name  pg_pin_name      voltage_name     pg_type    pg_pin_direction
……................
END PG_TO_VOLTAGE_MAP
```

The following example shows the PG pin mapped to its voltage in the `PG_TO_VOLTAGE_MAP` section:

```
#
BEGIN PG_TO_VOLTAGE_MAP
#
cell  pg_pin  voltage_name   pg_type        direction
LS_1  VDD1    VDD1           primary_power       -
^     VSS1    VSS            primary_ground      -
END PG_TO_VOLTAGE_MAP
```

Note:

The `direction` field is optional in the `PG_TO_VOLTAGE_MAP` section. If it is not specified, the `direction` default is `inout`.

Voltage Name to Value Mapping Section

You can map the voltage name to a voltage value in the `VOLTAGE_MAP` section, as shown:

```
BEGIN VOLTAGE_MAP
voltage_name   voltage_value
voltage_name   voltage_value
……................
END VOLTAGE_MAP
```

The following example shows the voltage name mapped to the voltage value:

```
BEGIN VOLTAGE_MAP
voltage_name    voltage_value
VDD1                 3.0
VDD2                 3.1
VSS                  0.0
END VOLTAGE_MAP
```

power_down_function Mapping Section

You can specify `power_down_function` and output/inout signal pin mapping in the `POWER_DOWN_FUNCTION_MAP` section, as shown:

```
BEGIN POWER_DOWN_FUNCTION_MAP
cell           pin          power_down_function
cell_name  signal_pin_name "power_down_function_name"
……................
END POWER_DOWN_FUNCTION_MAP
```

The `POWER_DOWN_FUNCTION_MAP` section is optional for libraries with single power and ground rail cells only. If the section is not specified, the default value is derived from the following Boolean expression:

```
" !primary_power pg_pin + primary_ground pg_pin "
```

The value is then added to the generated PG pin library.

The `POWER_DOWN_FUNCTION_MAP` section, as shown in the following example, is mandatory for libraries with multiple power and ground rail cells:

```
#
BEGIN POWER_DOWN_FUNCTION_MAP
#
cell       pin              power_down_function
LS_1       Z                "!VDD1 + !VDD2 +VSS"
LS_2       Z                "!VDD1 + !VDD2+ VSS1 + VSS2"
END POWER_DOWN_FUNCTION_MAP
```

Power Management Attribute Mapping Section

The `POWER_MANAGEMENT_ATTRIBUTE_MAP` section adds power management (PM) cell attributes to the PG map file. The `add_pg_pin_to_lib` command adds power management attributes to the generated PG pin library automatically if the data is provided in the PG map file. You should specify this PG attribute data in the following format:

```
BEGIN  POWER_MANAGEMENT_ATTRIBUTE_MAP
cell       (pg_)pin  attribute   value
cell_name  pin_name  attr_name   attr_value
END  POWER_MANAGEMENT_ATTRIBUTE_MAP
```

where `pin_name` is either the signal pin name or a valid PG pin name for the cell and attribute type. A hyphen (-) denotes a cell-level attribute not associated with any signal or PG pin for the cell.

The following PM attributes are available for all power management cells:

```
BEGIN  POWER_MANAGEMENT_ATTRIBUTE_MAP
cell       (pg_)pin   attribute                value
#LEVEL_SHIFTER cells
LS              -     is_level_shifter          true
^               -     level_shifter_type        LH/HL/LH_HL
^               -     input_voltage_range       (low , high)
^               -     output_voltage_range      (low , high)
LS              VDD   std_cell_main_rail        true
^               A     level_shifter_data_pin    true
^               EN    level_shifter_enable_pin  true
^               A     input_voltage_range       (low , high)
^               Y     output_voltage_range      (low , high)
#Coarse Grain SWITCH cells
SWITCH          -     switch_cell_type         coarse_grain
^              VDD   direction                input
^             sleep  switch_pin               true
^             VVDD   switch_function          !sleep
^             VVDD   pg_function               VDD
^             VVDD   direction                 output
#ALWAYS ON cells
AO              -     always_on                 true
^              A     always_on                 true
#ISOLATION cells
ISO             -     is_isolation_cell         true
^              A     isolation_cell_data_pin   true
^              EN    isolation_cell_enable_pin true
#RETENTION cells
RET_DFF         -     retention_cell            my_retention_cell
^            SAVE    retention_pin     (save/restore/save_restore, "0/1")
END  POWER_MANAGEMENT_ATTRIBUTE_MAP
```

Note:

The names under the `cell` and `(pg_)pin` columns, such as `VDD`, `sleep`, and so on, are dummy names for demonstration purposes.

The section lines that begin with the hash (#) character are comment lines, and the asterisk (*), hyphen (-), and caret (^) characters are wildcards. Wildcards and partial cell names, such as `CELL*/PIN*` for cell or pin names, are supported. In the second column, `(pg_)pin` specifies signal pins and PG pins. A hyphen (-) denotes a cell-level attribute.

The `is_level_shifter : true` and `is_isolation_cell : true` values are automatically added to the level-shifter and isolation cells if they are not in the input library file. The `add_pg_pin_to_lib` command recognizes the cell type through user-specified attributes in the PG map file.

If power_gating_cell and power_gating_pin syntax is included in the input library, the power_gating_cell and power_gating_pin values are automatically converted to retention_cell and retention_pin for modeling retention cells. Therefore, adding a map section for them is not necessary unless you want to explicitly specify a cell as a retention cell. In other words, if a retention cell is included in the input library but there is no power_gating_cell or power_gating_pin attribute, you should explicitly specify it in the map file.

The PG pin direction of a switch cell can be specified either in the POWER_MANAGEMENT_ATTRIBUTE_MAP section or in the PG_TO_VOLTAGE_MAP section. The POWER_MANAGEMENT_ATTRIBUTE_MAP section takes higher priority.

If you specify a power management attribute in the map file but it already exists in the input library, the specified information is ignored.

Support for macro switch cells is available in the add_pg_pin_to_lib command beginning with the B-2008.09-SP3 release.

Usage Model Recommendations

You should consider the recommendations described in the following sections when you are using the add_pg_pin_to_lib command.

Library With Complete Milkyway Library

If you are using a Milkyway library that covers all the cells that are included in the input non-PG library file, you can use the following usage models:

Library With a Single Power and Ground Rail

You do not need to specify the -pg_map_file option for libraries that contain cells with a single power and ground rail only. In this case, you should specify the -mw_library_name option to derive the mapping from the Milkyway FRAM view, as shown in the following example:

```
command_shell> add_pg_pin_to_lib non_pg.lib -mw_library_name {mw_lib}
-output pg.lib
```

In the example, the non_pg.lib file is a single rail logical library that is not based on PG pin syntax; mw_lib is the Milkyway library for non_pg.lib; and pg.lib is the name of the PG pin syntax library that is generated if the command runs successfully.

Library With Single and Multiple Power and Ground Rail Cells

The `-pg_map_file` option is required for libraries that contain cells with single and multiple power and ground rails. Be sure to specify all the sections in the PG map file. You can run the following command line options:

```
command_shell> add_pg_pin_to_lib non_pg.lib \
-mw_library_name {mw_lib} \
-pg_map_file pg.map -output pg.lib
```

In the example, `pg.map` is the PG map file name.

Library With a Partial Milkyway Library

If you are using a Milkyway library that does not cover all the cells that are included in the input non-PG library file, you should specify `-pg_map_file` for the cells that are missing in the Milkyway library, as shown in the following example:

```
command_shell> add_pg_pin_to_lib non_pg.lib \
-mw_library_name {mw_lib} \
-pg_map_file pg.map -output pg.lib
```

To determine whether a Milkyway FRAM library has single or multiple power and ground rails, use the `check_library -phys_property {cell}` library preparation command, as shown:

```
command_shell> set_check_library_options -reset
command_shell> set_check_library_options -phys_property cell
command_shell> check_library -mw_library_name XYZ
Level_Shifter StdCell   Z            Output         signal
                        A             Input          signal
                        I             Input          signal
                        VSS           Input/Output   ground
                        VDDL          Input/Output   power
                        VDDH          Input/Output   power
```

For more information about `check_library`, see the `check_library` man pages.

If a single power and ground cell is included in both the map file and in the Milkyway library, the names of the PG pins that are derived from the Milkyway FRAM view take precedence over the names you specify in the PG map file.

Library Without Milkyway Views

If you have a logical library only, you must specify the `-pg_map_file` option, as shown in the following example, to make the conversion for both a single power and ground library or a multiple power and ground library:

```
command_shell> add_pg_pin_to_lib non_pg.lib -pg_map_file pg.map \
-output pg.lib
```

Map Template Generation

The `add_pg_pin_to_lib` command can generate a PG map file template with the following sections: `PG_PIN_MAP`, `PG_TO_VOLTAGE_MAP`, `VOLTAGE_MAP`, `POWER_DOWN_FUNCTION_MAP`, and `POWER_MANAGEMENT_ATTRIBUTE_MAP`.

When generating the `PG_PIN_MAP` section, the `add_pg_pin_to_lib` command can derive the data from the input non-PG library and the Milkyway FRAM view to identify the signal to the PG pin associations. If a `rail_connection` complex attribute was specified on a few or all cells in the input library, the `rail_connection` column specifies a value in the `PG_PIN_MAP` section, as shown in the following example. The names under the `pg_pin` column are derived from the logical and the physical library information for each cell in the library.

```
BEGIN PG_PIN_MAP
cell    pin     rail_connection pg_pin
# cell 1: AO_buf
AO_buf  A               VDDB    VDDB?
AO_buf  A               -       VSS?
AO_buf  Y               VDDB    VDDB?
AO_buf  Y               -       VSS?
AO_buf  -               VDD     VDD
…
END PG_PIN_MAP
```

When the `PG_TO_VOLTAGE_MAP` section is generated, the cell's PG pins are mapped to the rails specified at the library level by the `voltage_map` complex attribute. The names of the rails are specified under the `voltage_name` column, as shown in the following example. If the cell is a switch cell, the `pg_pin` direction is included under the `direction` column.

```
BEGIN PG_TO_VOLTAGE_MAP
cell    pg_pin      voltage_name    pg_type        direction
# cell 1: AO_buf
AO_buf  VDD?            VDD?    primary_power
AO_buf  VDDB?           VDDB?   backup_power
AO_buf  VSS?            VSS     primary_ground
…
END PG_TO_VOLTAGE_MAP
```

The rails' voltage values are specified when the `VOLTAGE_MAP` section is generated, as shown in the following example. If ground pins from the Milkyway library are not included in the logical library, they are added with the value set to `0`.

```
BEGIN VOLTAGE_MAP
voltage_name    voltage_value
VSS1                    0?
VSS                     0?
VVDD                    0.7?
VDDB                    0.6?
VDD2                    1.2?
```

```
VDD1                    0.8?
VDD                     0.8?
END VOLTAGE_MAP
```

All other voltage values are derived from the `operating_conditions` group in the input logical library for single rail cells and from the `power_supply` group for multiple-rail PG pins.

When the `POWER_DOWN_FUNCTION_MAP` section is generated, the output pins for each cell are listed under the `pin` section. The `power_down_function` Boolean expressions for the PG pins are specified under the `power_down_function` column, as shown in the following example.

```
BEGIN POWER_DOWN_FUNCTION_MAP
cell    pin           power_down_function
CELL1    *               !VDDB?+VSS?
CELL2    *               !VDD1?+!VDD2?+VSS?
CELL3    *               !VDD1?+!VDD2?+VSS?
 …
END POWER_DOWN_FUNCTION_MAP
```

If the cell includes Power Management attributes in the input non-PG library, the `add_pg_pin_to_lib` command extracts the attributes and adds them to the Power Management section in the generated map template file. For example, if the cell includes an `is_level_shifter` attribute in cell `LS1` with a value of `true`, the following line is added to the Power Management map section:

```
LS1  -  is_level_shifter    true
```

If the cell contains old Power Management attributes, such as `power_gating_cell` and `power_gating_pin`, the command converts them to `retention_cell` and `retention_pin`, respectively, and adds the converted names to the Power Management map section.

Table 1-1 summarizes the translation between the old `power_gating_cell` and `power_gating_pin` attributes and their new equivalents.

Table 1-1 Retention Modeling Syntax Changes

Old Syntax	New Syntax
`power_gating_cell : cell_type;`	`retention_cell : cell_type;`
Simple attribute	Simple attribute
`power_gating_pin (power_pin_[1-5], " 0 " | " 1 ");`	`retention_pin (pin_class, disable_value);`
Complex attribute	Complex attribute

After the Power Management attributes are extracted and listed in the map template file, you add the rest of the Power Management attributes to make a complete Power Management attribute set for the cell. The `add_pg_pin_to_lib` command adds the following information to the generated template PG map file for level-shifter cells if the information is missing in the original input library:

```
# For Level shifter cells
LS   -     level_shifter_type       LH?
LS   VDD?  std_cell_main_rail       true
LS   A?    level_shifter_data_pin   true
LS   EN?   level_shifter_enable_pin true
LS   -     output_voltage_range     (low_value?, high_value?)
LS   -     input_voltage_range      (low_value?, high_value?)
```

where `VDD` is a primary power PG pin in the cell, `A` is a signal input pin, and `EN` is the second signal input pin, if any.

The `add_pg_pin_to_lib` command adds the following information to the generated template PG map file for switch cells if the information is missing in the original input library:

```
# For Switch cells
SWITCH   -      switch_cell_type  coarse_grain
SWITCH  sleep?  switch_pin        true
SWITCH  VVDD?   switch_function   sleep?
SWITCH  VVDD?   pg_function       VDD?
```

where `VVDD` is a virtual type of PG pin and the internal power for the switch cell, `VDD` is a primary power PG pin, and `sleep` is a signal input switch pin of the coarse-grain switch cell named `SWITCH`. Because the cell is a switch cell, the `PG_TO_VOLTAGE_MAP` section is also updated under the `pg_type` and `direction` columns.

The `VDD` direction for `pg_function` is `input`, and the PG pin type is `primary_power`. The `VVDD` direction is `output`, and the `pg_type` is `internal_power`.

The `add_pg_pin_to_lib` command adds the following information to the generated template PG map file for isolation cells if the information is missing in the original input library:

```
# For Isolation cells
ISO  A?    level_shifter_data_pin  true
ISO  EN?   level_shifter_enable_pin true
```

The question mark (?) character is appended to the value of pins `A` and `EN` because you must confirm the type of input pins on the isolation cell.

Note:

Make sure the map template file is clear of all question mark (?) characters so that a final version of the PG map file can be used for the final conversion phases.

It is recommended that you add the back-bias syntax as described in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide* in the PG map file if it is applicable. Additional options are available under the `PG_TO_VOLTAGE_MAP` sections to add the back-bias attributes to the library. All other sections of the PG map file support the back-bias syntax if it is applicable, for example:

```
BEGIN PG_TO_VOLTAGE_MAP
cell       pg_pin  voltage_name  pg_type           direction
physical_connection  related_bias_pin
# cell 1: BIAS_CELL
BIAS_CELL  vss     vss           primary_ground     -
-                    -
^          vdd     vdd           primary_power      -
-                    vdds
^          vdds    vdds          nwell              internal
device_layer         -
…
```

Recommended Flows

The following flows and usage models are supported by `add_pg_pin_to_lib`.

Complete Access to Milkyway Library FRAM Views

If all cells have a single power and ground rail (one power and one ground rail), the `-pg_map_file` option is not required. However, you can specify `-mw_library_name` to derive the mapping from the Milkyway FRAM view.

Note:

If any cell has more than one power and ground rail, the PG map file is required for conversion.

No Access to Milkyway Library FRAM Views (Logical Library Only)

If you specify a non-PG logical library only, even if it is a single power and ground rail library or a multiple-rail library, you must specify the `-pg_map_file` option to complete the conversion successfully.

Access to a Partial Milkyway Library Only

If some cells exist in both the logical library and the physical Milkyway library while others exist only in the logical library, you must specify the PG map file for the cells that are missing in the Milkyway library FRAM views and for all cells that have more than one pair of PG pins.

Syntax Changes in Libraries Converted to PG Pin Libraries

Table 1-2 shows the most common syntax changes that occur in the library view when logical libraries without PG pin syntax are converted to libraries with PG pin syntax:

Table 1-2 Syntax Changes in Libraries Converted to PG Pin Libraries

Level	Attribute Name and syntax	Action	Comment
Library level	`add_to_pg_pin_lib : true`	Added	This is a user-defined, library-level attribute.
	`power_supply ( [power_supply_name] ) {default_power_rail : string; power_rail( string, float ); }`	Deleted	This library-level group is the old non-PG pin syntax.
	`voltage_map( string, float );`	Added	This library-level attribute replaces the `power_supply` group and is controlled by the `voltage_map` section of the PG map file.
Cell level	`rail_connection (string, float);`	Deleted	This cell-level attribute is the old non-PG pin syntax.
Cell level (cont.)	`leakage_power ([leakage_power_name]) { /* Attributes */ power_level : string; /* delete */ related_pg_pin : string; value : float (>0.0); when : virtual_attribute; /* Sub-groups */ power (power_name) { }`	Updated	All `power_level` attributes are replaced with the `related_pg_pin` attributes.
	`pg_pin (pg_pin_name) { pg_type:enum (internal_power, backup_power, backup_ground, or primary_power); voltage_name: string; direction: input/output/inout/internal physical_connection: enum (device_layer, routing_pin); related_bias_pin :string; user_pgtype:string; }`	Added	PG pin syntax is added to every cell, and it replaces the `rail_connection` syntax in certain cases.

Table 1-2 Syntax Changes in Libraries Converted to PG Pin Libraries (Continued)

Level	Attribute Name and syntax	Action	Comment
Pin level	`internal_power ([internal_power_name]) { /* Attributes */ power_level: string; /* delete */ related_pg_pin: string; when: virtual_attribute; /* Functions */ values ( <unknown_arugments> ); /* Sub-groups */ ... }`	Updated	All `power_level` attributes are replaced with the `related_pg_pin` attributes.
	`related_ground_pin : string; related_power_pin : string;`	Added	All signal pins are associated with the PG pins. These are driven by the specification inside the `pg_pin_map` section.
	`power_down_function : string`	Added	This information is added to all output signal pins and is driven through the `power_down_function_map` section.

Note:

Additional power management attributes are added to the library based on information you specify inside the `power_management_map` section of the PG map file. These can be cell-level, pin- level, or PG-pin level attributes.

Validating PG Pin Libraries

The library validation described in the following sections checks for UPF compliance.

Validating Library Differences After Conversion

Because the generated logical library is an ASCII file, you can use the `diff` command to determine the difference between the non-PG and the generated PG library. This allows you to validate that the only difference between the non-PG source library and the PG library is the addition or change of PG syntax and to ensure that no other data, such as characterization information, has changed in the PG library.

Checking for Errors and Warnings

Use the `read_lib` command, as shown, to confirm that no critical Library Compiler warning or error messages were generated on the PG library after running the `add_pg_pin_to_lib` command:

```
command_shell> read_lib pg.lib
```

If the library did not compile successfully, check the specific error and fix the problem as necessary. In rare circumstances, `add_pg_pin_to_lib` generates the library successfully but Library Compiler fails to compile the library successfully. This can occur in the following circumstances:

- A bug in the command
- A Library Compiler check introduced in the PG pin syntax
- User bypass of all command checks that add unknown Liberty syntax that is not supported in Library Compiler

Reporting PG Pin Attributes

When a library has successfully been read with the `read_lib` command, you can use the `-pg_pin` option with the `report_lib` command to report all the PG pin attributes, as shown:

```
command_shell> report_lib -pg_pin library_name
```

The following is an excerpt from a report for a non-PG pin based library:

```
…
Name : my_cell RAIL CONNECTION(VDD1): VDD1
RAIL CONNECTION(VDD2): VDD2
CELL(my_cell): 1, ;
PIN(A): in, 1, , , ;
INPUT_SIGNAL_LEVEL(A): VDD1
END_PIN A;
PIN(Z): out, 0, , , , , , ;
OUTPUT_SIGNAL_LEVEL(Z): VDD2
END_PIN Z;
END_CELL LH_LS;
…
```

The following shows the report after converting to a library with PG pin syntax:

```
Name : my_cell
CELL(my_cell): 1, ;
PG_PIN(VDD1):
```

```
VOLTAGE_NAME: VDD1
PG_TYPE: primary_power
END_PG_PIN VDD1;
PG_PIN(VDD2):
VPG_TYPE: primary_power
VOLTAGE_NAME: VDD2
PG_TYPE: primary_power
END_PG_PIN VDD2;
PG_PIN(VSS):
VOLTAGE_NAME: VSS
PG_TYPE: primary_ground
END_PG_PIN VSS;
PIN(A): in, 1, , , ;
INPUT_SIGNAL_LEVEL(A): VDD1
RELATED_POWER_PIN : VDD1
RELATED_GROUND_PIN : VDD1
END_PIN A;
PIN(Z): out, 0, , , , , , ;
RELATED_POWER_PIN : VDD2
RELATED_GROUND_PIN : VDD2
END_PIN Z;
END_CELL LH_LS;
```

Library Checking With the check_library Command

The `check_library` command checks the integrity of the individual logical and physical libraries. It compares logical libraries, compares logical and physical libraries, and checks that the intraphysical libraries and technology files are consistent.

Logical Versus Physical View Checks

The `check_library` command has been available in Design Compiler Topographical and IC Compiler beginning with the A-2007.12 release. The `check_library` command performs data consistency checking between logical and physical libraries, as shown:

```
command_shell> check_library -mw_library_name $path/xyz
-logic_library_name
xyz_pgpin.db
```

where `xyz` is the name of the Milkyway library where the FRAM views of the library are located.

You can also use the `check_library` command to generate the details of the physical library cell information in the Milkyway library. Use the following set of commands to enable reporting of the PG pins and signal pins of any physical Milkyway library:

```
command_shell> set_check_library_options -reset
command_shell> set_check_library_options -phys_property cell
command_shell> check_library -mw_library_name xyz
```

```
…
Level_Shifter StdCell Z Output signal
                      A Input signal
                      I Input signal
                     VSS Input/Output ground
                     VDDL Input/Output power
                     VDDH Input/Output power
```

Logical Library Checks for UPF Compliance

In the B-2008.09 release, `check_library` was enhanced to check for discrepancies between two or more logical libraries.

You can use the `-upf` option with the `check_library` command, as shown, to make sure that the only difference between the non-PG logical libraries and the PG logical libraries is the PG pin syntax and that no other change to any other data in the libraries has occurred:

```
command_shell> set_check_library_options  -upf
command_shell> check_library -logic_library_name {non_pg_lib.db

pg_lib.db}
```

The report states all possible discrepancies between the two libraries, such as discrepancies in the following areas:

- Library-level information:
  - Low-power attribute mismatches
  - Operating conditions mismatches
  - Cell classification mismatches (such as standard cells, switch cells, level-shifter cells, and so on)
- Cell-level mismatches:
  - Low-power attribute mismatches
  - pin group and pg_pin group mismatches
  - pin group and pg_pin lower power attribute mismatches
  - Timing arc mismatches

Library Checking Rules

The `add_pg_pin_to_lib` command checks to make sure that the following conditions are met for the power management attributes map section.

Checks for Level Shifter Cells

The `add_pg_pin_to_lib` command checks the following conditions for level-shifter cells:

- The `std_cell_main_rail` Boolean attribute must satisfy the following properties:
  - The attribute must be specified as the `related_power_pin` value for one of the level-shifter signal pins.
  - The attribute must be defined in a `primary_power` power pin.
- The data input and output pins cannot be related to the same power pin.
- The `input_voltage_range` and `output_voltage_range` attributes must always be defined together.
- The `lower_bound` value for the `input_voltage_range` and `output_voltage_range` attributes must be less than or equal to the `upper_bound` value.
- The `level_shifter_enable_pin` pin-level attribute can be defined only on an input pin.
- A level-shifter cell must satisfy one of the following conditions:
  - Have two pins, including one input pin and one output pin.
  - Have three pins, including one input pin, one `level_shifter_enable_pin` pin, and one output pin.
- The `input_signal_level` rail name of the data input pin and `output_signal_level` rail name of the output pin must be different.

Checks for Switch Cells

The `add_pg_pin_to_lib` command checks the following conditions for switch cells:

- The switch function can contain only switch pins whose input pin, set by the `switch_pin` attribute, is set to true.
- A coarse-grain switch cell must satisfy the following rules:
  - The `switch_cell_type` attribute must be set to the `coarse_grain` value.

  - It must define the cell-level `switch_function` attribute to identify the control logic of its switch pins.
  - It must have at least one switch pin.
  - It must have at least one controlled power and ground pin and one regular power and ground pin: that is, a virtual VSS `pg_pin` and a VSS `pg_pin`, or a virtual VDD `pg_pin` and a VDD `pg_pin`. Each of these output power pins must have a `pg_function` Boolean expression containing input power pins.
- The `pg_function` attribute must contain only the input power and ground pins.
- If the cell is a macro switch cell, the `switch_cell_type` attribute value must be `fine_grain`.

Checks for Always-On Cells

The `add_pg_pin_to_lib` command checks the following conditions for always-on cells:

- Always-on cells must have a secondary power pin as a backup.
- All signal pins, including all always-on pins and non always-on pins, must be related to the backup PG pin.

Checks for Isolation Cells

The `add_pg_pin_to_lib` command checks the following conditions for isolation cells:

- The `isolation_cell_data_pin` and `isolation_cell_enable_pin` pin-level attributes can only be specified on an input pin.
- An isolation cell must have three pins, including one input data pin, one `isolation_cell_enable_pin` pin, and one output pin.

Checks for Retention Cells

The `add_pg_pin_to_lib` command checks the following conditions for retention cells:

- The `retention_pin` attribute must be defined on an input pin.
- The `retention_cell` and `retention_pin` attributes should always be specified together in any retention cell.

Limitations

The `add_pg_pin_to_lib` command has the following limitation: Some legacy libraries use signal pins through the use of pin groups in Liberty to represent power and ground PG pins, also called fake PG pins. These libraries were created before PG pin syntax was introduced to Liberty. The `add_pg_pin_to_lib` command does not support these input libraries for conversion and will issue an error message if they are identified in any library.

References

For information about Liberty syntax for modeling the following types of cells, see the "Advanced Low-Power Modeling" chapter in the *Library Compiler Modeling Timing, Signal Integrity, and Power in Technology Libraries User Guide*:

- PG pin cells
- Level-shifter and isolation cells
- Switch cells
- Retention cells
- Always-on cells

2

Updating Secondary PG Pins Application Note

This application note describes how to use the `update_mw_port_by_db` command to update the port type of secondary power and ground (PG) pins in a FRAM-based view with information from a .db library. In many cases, the FRAM view in the library is missing secondary PG pin information. The port type must be updated to allow Milkyway tools to complete routing. This application note contains the following sections:

PG Pin Types

There are two types of PG pins: primary and secondary. To define the type of PG pin in the .db library, you should use the `pg_type` attribute, which is used to update the FRAM view to match the .db library. The values for the `pg_type` attribute of a primary PG pin are

- primary_power
- primary_ground

If there are multiple primary_power pins in a cell, as in a level shifter, set the `std_cell_main_rail` attribute to true to designate the pin as a primary power pin, as shown in the following example:

```
cell (<cell_name>) {
…
pg_pin (<pg_pin_name>) {
…
pg_type : <pg_type>;
std_cell_main_rail : boolean;
}
}
```

The values for the `pg_type` attribute of a secondary PG pin are

- backup_power
- backup_ground
- internal_power
- internal_ground

If there are multiple primary_power pins in a cell and the `std_cell_main_rail` attribute is set to either false or not defined as true, the pins are regarded as secondary PG pins.

Mapping Secondary PG Port Types from .db to FRAM

In Milkyway tools, the `update_mw_port_by_db` command maps the PG type of the secondary PG pins from .db to FRAM. If there is a secondary PG pin in the input .db file, the `update_me_port_by_db` updates the FRAM port type of the PG pin, according to the following mapping table:

Table 2-1 Mapping Secondary PG Port Type from DB to FRAM

PG Pin Type in .db			GPortTable type	Query type
backup_power			Power BackupPG	Backup Power
backup_ground			Ground BackupPG	Backup Ground
internal_power			Power InternalPG	Backup Internal
internal_ground			Ground Internal PG	Backup Internal
primary_ground			Ground	Ground
primary_power	Only one primary power in cell		Power	Power
	Multiple primary power in cell	Std_cell_main_rail true	Power	Power
		Std_cell_main_rail false	Power BackupPG	Backup Power

The syntax to use the `update_mw_port_by_db` command is

```
update_mw_port_by_db
-db_file {db_file_name_list}
-mw_lib mw_library_name
```

The arguments for the `update_mw_port_by_db` command are

`-db_file {`*`db_file_name_list`*`}`

This is a required argument. `db_file_name_list` is a list that specifies one or more valid .db file names.

`-mw_lib <mw_library_name>`

This is a required argument. `mw_library_name` is a string that specifies a single valid Milkyway library name.

The following example shows how to use the `update_mw_port_by_db` command when the Milkyway library is `test` and its cells can be found in the .db file `test.db`:

```
Milkyway> update_mw_port_by_db -db_file {test.db} -mw_lib test
```

Verifying That the PG Pins Were Updated Correctly

After you run the `update_mw_port_by_db` command, verify that the secondary PG pins were updated in the FRAM file correctly.

Get the secondary PG pin type from the .db file, using dc_shell by doing the following:

1. Start dc_shell.
2. Read the .db library using the `read_db` command.
3. Get the .db library name using the `list_lib` command.
4. Get the PG pin type information using the `report_lib -pg_pin` command.
5. Search for the terms *internal* and *backup* in the `pg_type` section of the pg_pin file generated by the `report_lib` command.

Get the associated PG pin port type in the FRAM view. This can be determined by using one of the following methods:

Milkyway Select or Query Method

1. Start Milkyway.
2. Open a library using the `geOpenLib` command.
3. Open a cell using the `geOpenCell` command.
4. Select a secondary PG pin using the `geNameSelect` command.
5. Query the secondary PG pin using the `geQueryObject` command.
6. Repeat steps 1 through 5 before and after using the `update_mw_port_by_db` command.
7. Compare the pg_type in the .db file with PORT_TYPE in FRAM for the same secondary PG pin.

Milkyway Dumping GPortTable Method

1. Start Milkyway.
2. Open a library using the `geOpenLib` command.
3. Write out the GPortTable using the `dbDumpGPortTable` command.
4. Repeat step 3 before and after using the `update_mw_port_by_db` command.
5. Compare the pg_type in the .db file with PORT_TYPE in FRAM for the same secondary PG pin.

Verifying That the PG Pins Were Updated Correctly - An Example

In this example, the input .db file is `test.db` and the Milkyway library is `test`.

Get the secondary PG pin type from the .db file:

1. Start dc_shell.
2. Read the .db library using the `read_db test.db` command.
3. Get the .db library name using the `list_lib` command. The command returns `test`.
4. Get the PG pin type information using the `report_lib -pg_pin test` command. The command returns all the PG pin types in the transcript.
5. Search for internal_power in the output file. The following example shows a sample output file:

```
CELL(HDRDID2HVT): 34.560001, r, switch_cg;
   LEAKAGE_POWER :
      WHEN : !NSLEEPIN1 !NSLEEPIN2
   PG_PIN(TVDD): in,
      VOLTAGE_NAME: TVDD
      PG_TYPE: primary_power
   END_PG_PIN TVDD;
PG_PIN(VDD): out,
VOLTAGE_NAME: VDD
PG_TYPE: internal_power
PG_FUNCTION: TVDD
SWITCH_FUNCTION: !NSLEEPIN1 + !NSLEEPIN2
END_PG_PIN VDD;
```

Look for the pin name associated with internal_power in PG_PIN. In the example provided in this section, the PG_PIN VDD in CELL (HDRDID2HVT) is a secondary PG pin.

Get the associated PG pin port type in the FRAM view. This verifies that the secondary PG pin PORT_TYPE in FRAM is updated after using the `update_mw_port_by_db` command. For this example, use the select/query method.

1. Start Milkyway.
2. Open the library using the `geOpenLib` command.
3. Open a cell using the `geOpenCell` command.
4. Select the secondary PG pin VDD using the `geNameSelect` command.
5. Query the secondary PG pin VDD using the `geQueryObject` command. The command returns the pin information. See the secondary PG pin VDD query output in following transcript.
6. Repeat steps 1 to 5 before and after using the `update_mw_port_by_db` command.

The result of the query before running the `update_mw_port_by_db` command is shown in the following example:

```
OBJECT TYPE : PIN [ID #x801]
PIN NAME : VDD
LAYER : M1 (31)
ACCESS_DIRECTION:
PIN_DIRECTION:
PORT_TYPE : Power
TOTAL POINTS : 38
BBOX : (0.0000 1.6350) (9.6000 1.9650)
BBOX : (3.3050 1.9650) (8.2100 2.0000)
```

The result of the query after running the `update_mw_port_by_db` command is shown in the following example:

```
OBJECT TYPE : PIN [ID #x801]
PIN NAME : VDD
LAYER : M1 (31)
ACCESS_DIRECTION:
PIN_DIRECTION:
PORT_TYPE : Internal Power
TOTAL POINTS : 38
BBOX : (0.0000 1.6350) (9.6000 1.9650)
BBOX : (3.3050 1.9650) (8.2100 2.0000)
```

The PORT_TYPE in the query result indicates that the VDD PORT_TYPE pin is Power before running the `update_mw_port_by_db` command, and it is *Internal_Power* after using the command. Repeat the verification procedure for all secondary pins.

Alternatively, to get the associated PG pin port type in the FRAM view using the GPortTable method, follow these steps:

1. Write out the GPortTable of the Milkyway library before and after, using the `update_mw_port_by_db` command.

   ```
   Milkyway> dbDumpGPortTable test pre_test.GPortTable
   Milkyway> dbDumpGPortTable test post_test.GPortTable
   ```

2. Compare the PG pin PORT_TYPE between the two tables. This can be done by using the `diff` command on the two GPortTable files in UNIX.

   ```
   diff pre_test.GPortTable post_test.GPortTable
   ```

   The following example shows results that indicate that the PORT_TYPE of the secondary PG pin TVDD has been updated from "Power" to "Power" "BackupPG".

   ```
   < dbSetCellPortTypes "test_org" "PTBUFFD1HVT" '(
   ---
   > dbSetCellPortTypes "test" "PTBUFFD1HVT" '(
   < ("TVDD" "Inout" "Power" )
   ---
   > ("TVDD" "Inout" "Power" "BackupPG" )
   ```